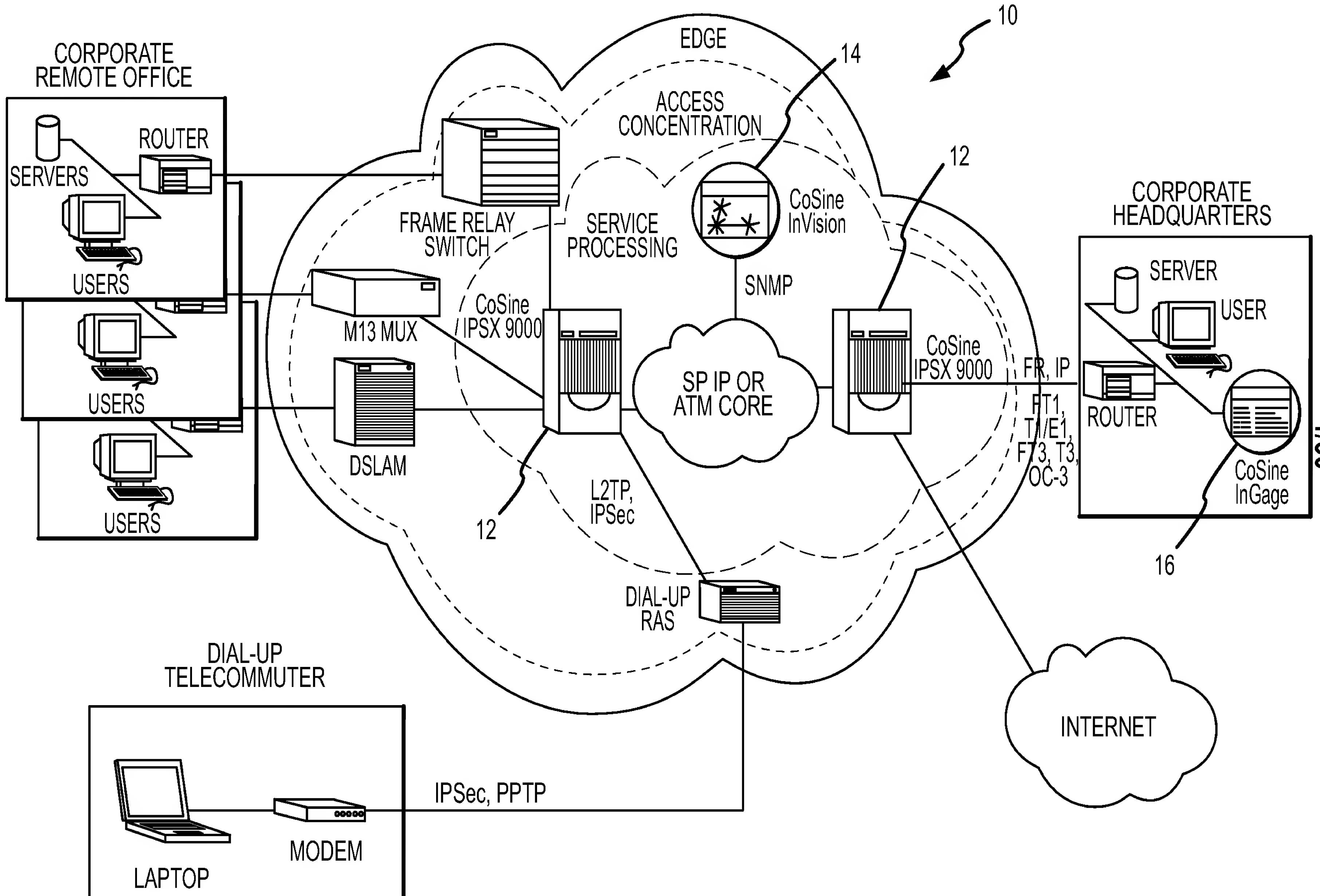

10
CORPORATE REMOTE OFFICE
SERVERS
ROUTER
USERS
USERS
USERS
EDGE
ACCESS CONCENTRATION
SERVICE PROCESSING
14
12
FRAME RELAY SWITCH
M13 MUX
DSLAM
CoSine IPSX 9000
CoSine InVision
SNMP
SP IP OR ATM CORE
CoSine IPSX 9000
12
L2TP, IPSec
DIAL-UP RAS
FR, IP
FT1, T1/E1, FT3, T3, OC-3
CORPORATE HEADQUARTERS
SERVER
USER
ROUTER
CoSine InGage
16
INTERNET
DIAL-UP TELECOMMUTER
LAPTOP
MODEM
IPSec, PPTP

FIG.1

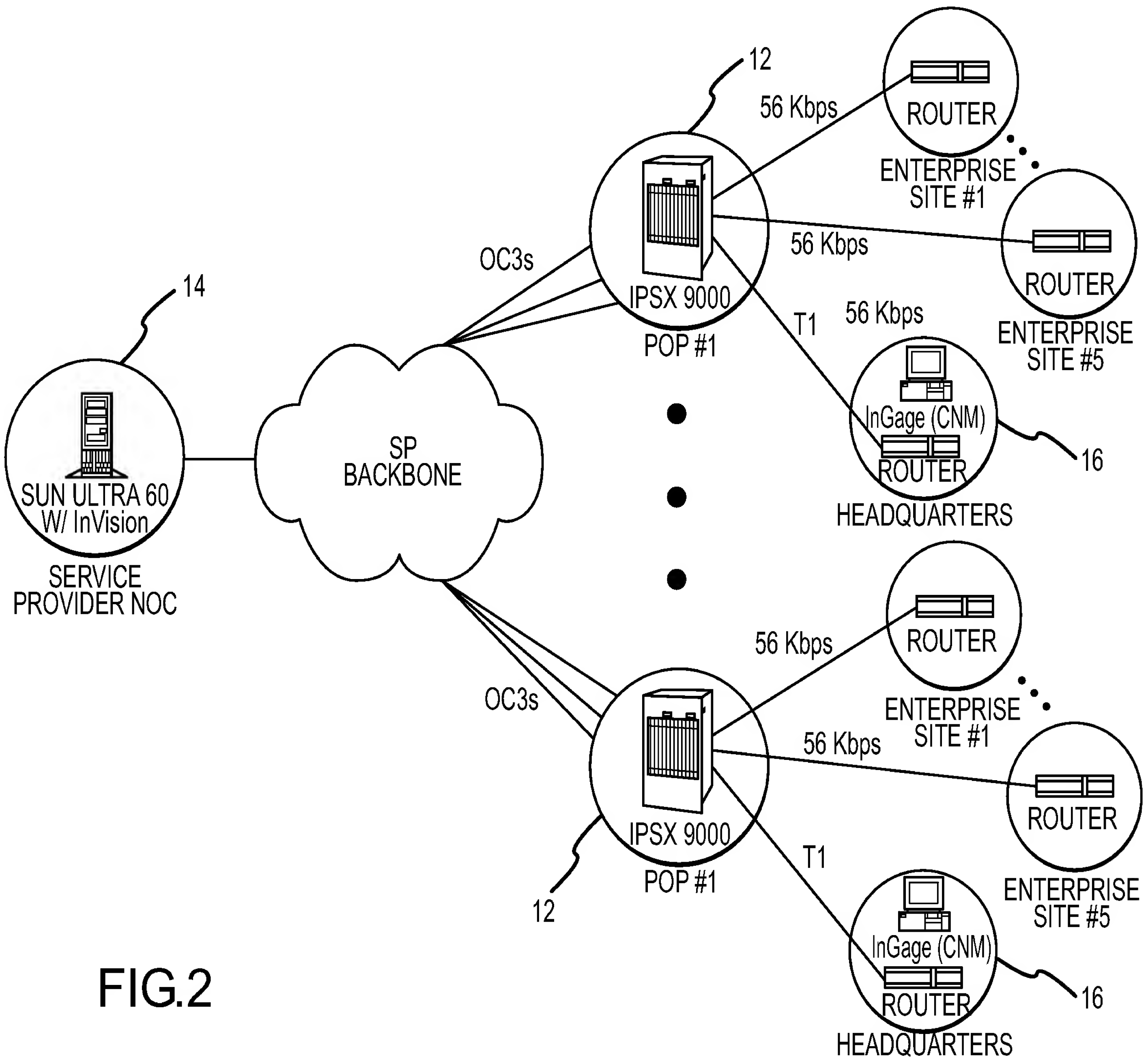

14
SUN ULTRA 60
W/ InVision
SERVICE
PROVIDER NOC
SP
BACKBONE
OC3s
12
IPSX 9000
POP #1
56 Kbps
ROUTER
ENTERPRISE
SITE #1
56 Kbps
ROUTER
ENTERPRISE
SITE #5
T1
56 Kbps
InGage (CNM)
ROUTER
HEADQUARTERS
16
OC3s
12
IPSX 9000
POP #1
56 Kbps
ROUTER
ENTERPRISE
SITE #1
56 Kbps
ROUTER
ENTERPRISE
SITE #5
T1
InGage (CNM)
ROUTER
HEADQUARTERS
16

FIG.2

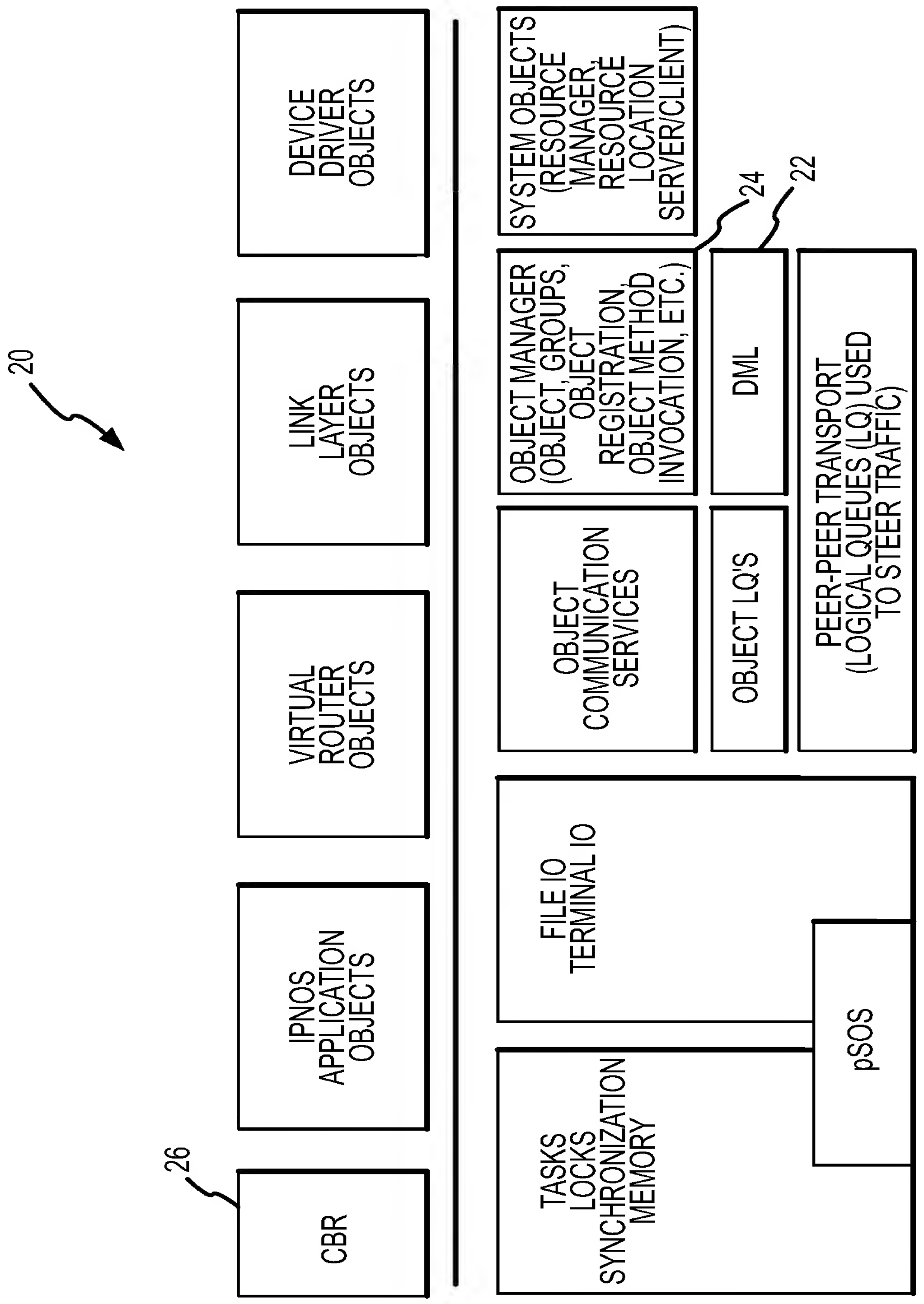

20
26
CBR
IPNOS APPLICATION OBJECTS
VIRTUAL ROUTER OBJECTS
LINK LAYER OBJECTS
DEVICE DRIVER OBJECTS
TASKS LOCKS SYNCHRONIZATION MEMORY
FILE IO TERMINAL IO
pSOS
OBJECT COMMUNICATION SERVICES
OBJECT MANAGER (OBJECT, GROUPS, OBJECT REGISTRATION, OBJECT METHOD INVOCATION, ETC.)
24
SYSTEM OBJECTS (RESOURCE MANAGER, RESOURCE LOCATION SERVER/CLIENT)
OBJECT LQ'S
DML
22
PEER-PEER TRANSPORT (LOGICAL QUEUES (LQ) USED TO STEER TRAFFIC)

FIG.3

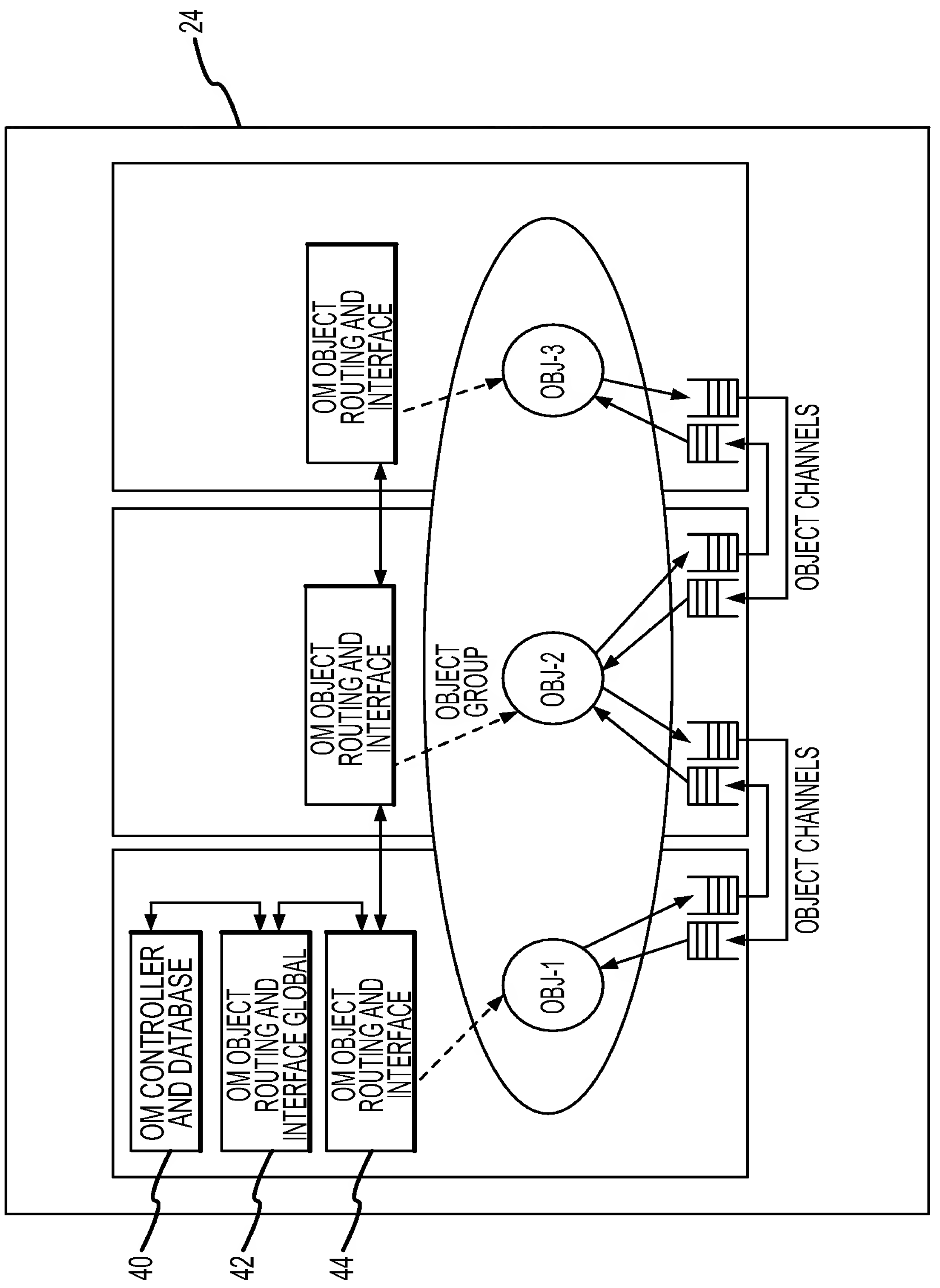

24
OM CONTROLLER AND DATABASE
OM OBJECT ROUTING AND INTERFACE GLOBAL
OM OBJECT ROUTING AND INTERFACE
OM OBJECT ROUTING AND INTERFACE
OM OBJECT ROUTING AND INTERFACE
OBJ-1
OBJ-2
OBJ-3
OBJECT GROUP
OBJECT CHANNELS
OBJECT CHANNELS
40
42
44

FIG.4

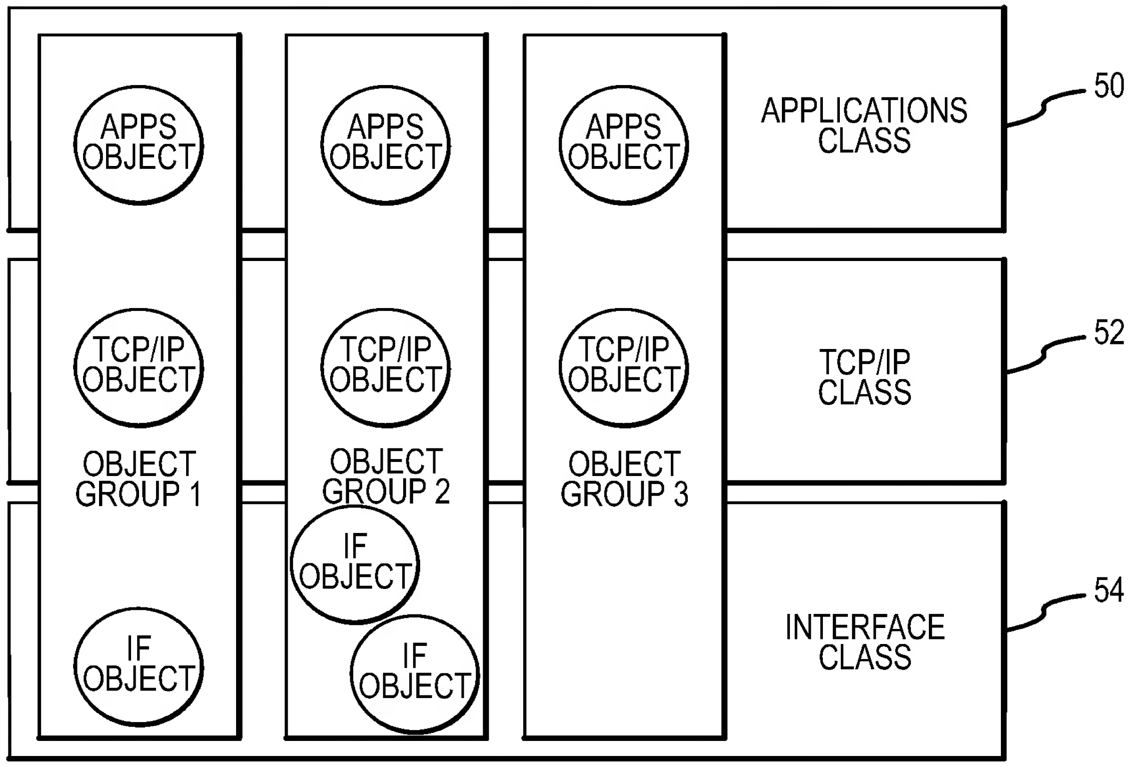

APPS OBJECT
APPS OBJECT
APPS OBJECT
APPLICATIONS CLASS
50
TCP/IP OBJECT
TCP/IP OBJECT
TCP/IP OBJECT
TCP/IP CLASS
52
OBJECT GROUP 1
OBJECT GROUP 2
OBJECT GROUP 3
IF OBJECT
IF OBJECT
IF OBJECT
INTERFACE CLASS
54

FIG.5

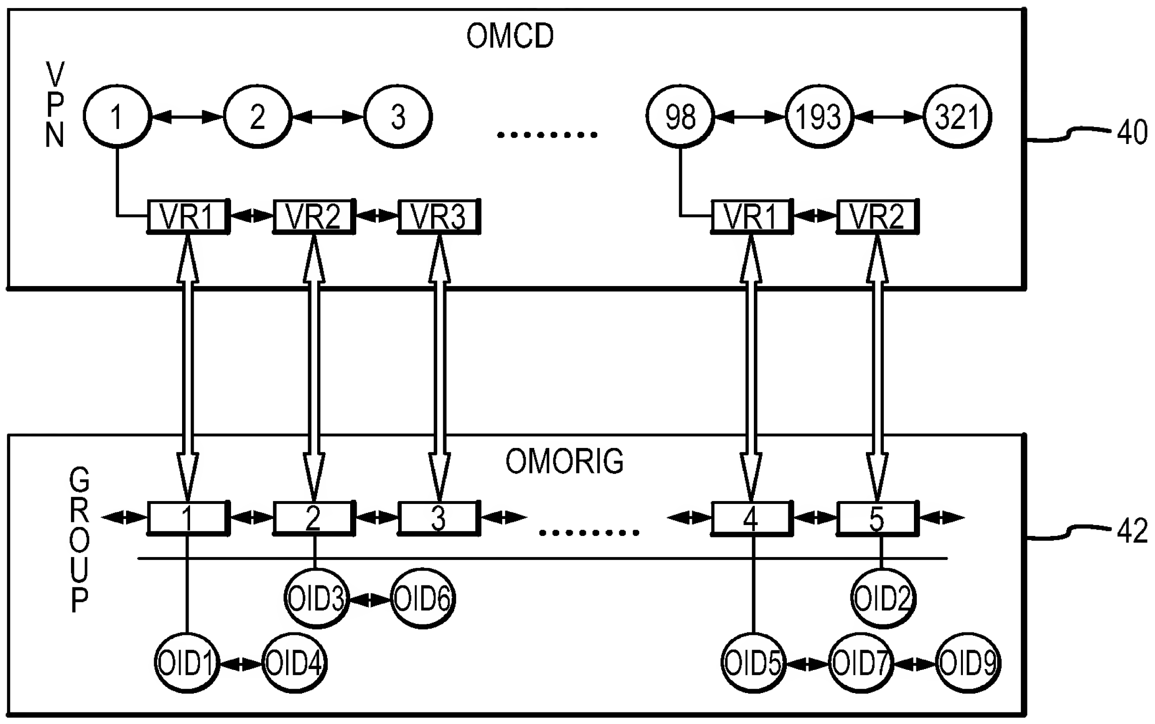

OMCD
VPN
1
2
3
....
98
193
321
VR1
VR2
VR3
VR1
VR2
40
OMORIG
GROUP
1
2
3
....
4
5
OID3
OID6
OID2
OID1
OID4
OID5
OID7
OID9
42

FIG.6

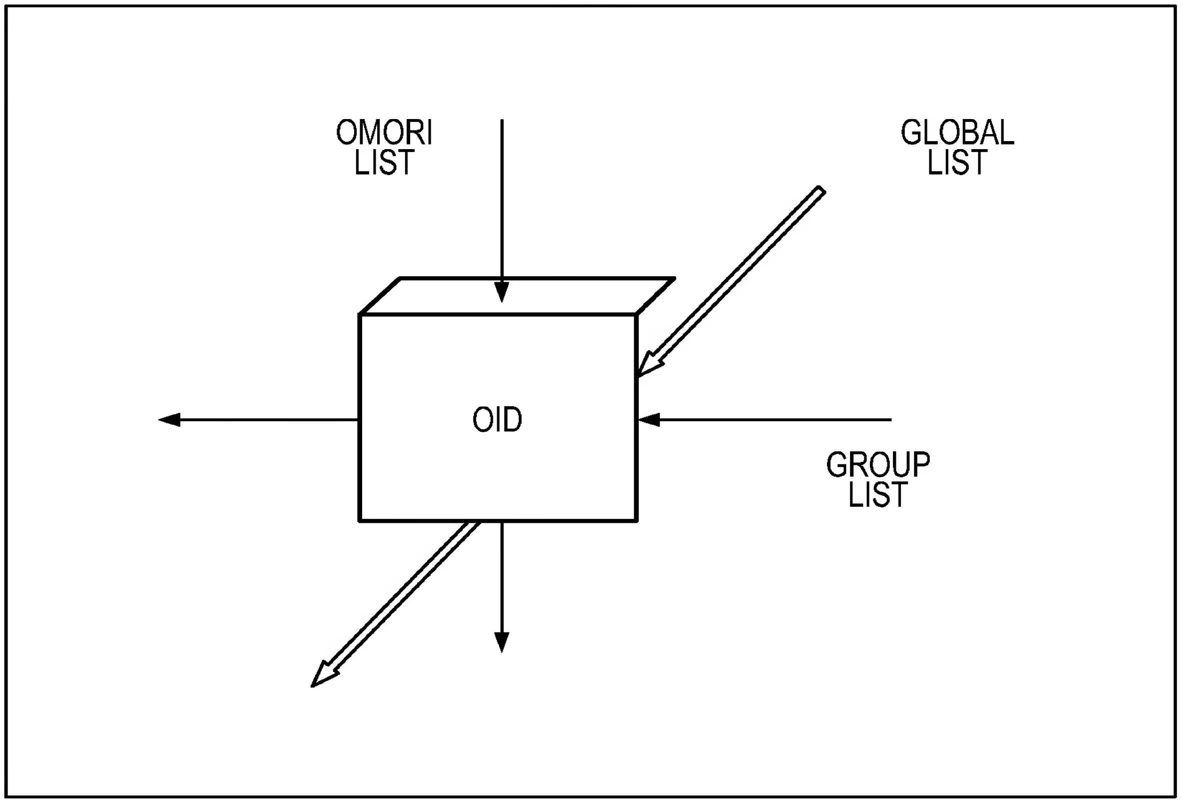


FIG.7

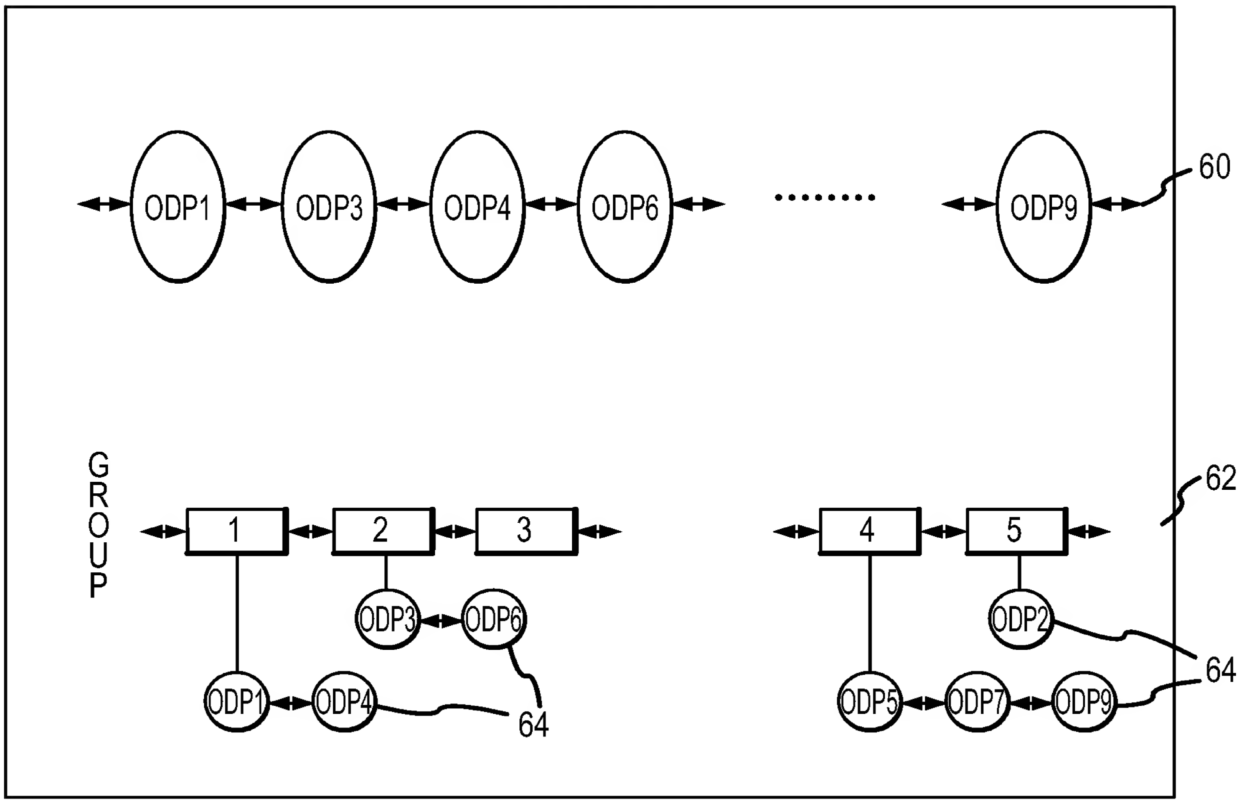

ODP1
ODP3
ODP4
ODP6
ODP9
60
GROUP
1
2
3
4
5
62
ODP3
ODP6
ODP2
ODP1
ODP4
64
ODP5
ODP7
ODP9
64

FIG.8

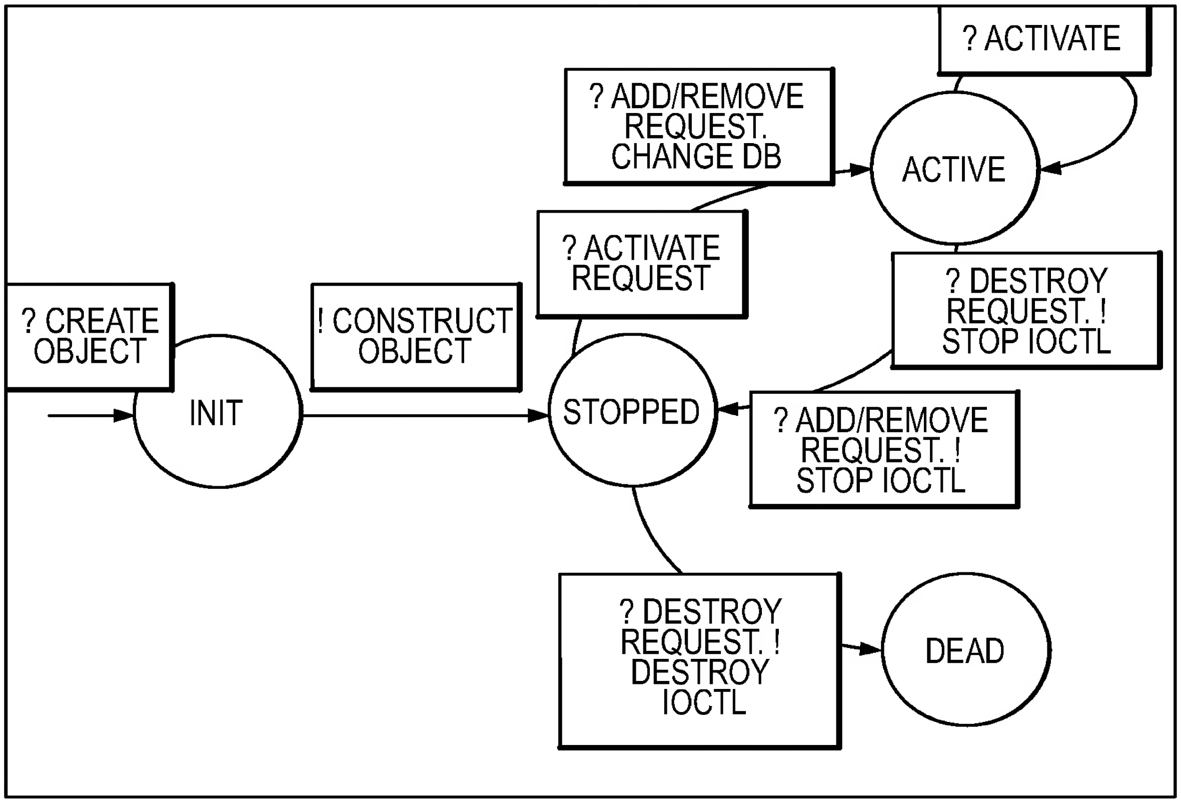

? CREATE OBJECT
INIT
! CONSTRUCT OBJECT
STOPPED
? ACTIVATE REQUEST
? ADD/REMOVE REQUEST. CHANGE DB
ACTIVE
? ACTIVATE
? DESTROY REQUEST. ! STOP IOCTL
? ADD/REMOVE REQUEST. ! STOP IOCTL
? DESTROY REQUEST. ! DESTROY IOCTL
DEAD

FIG.9

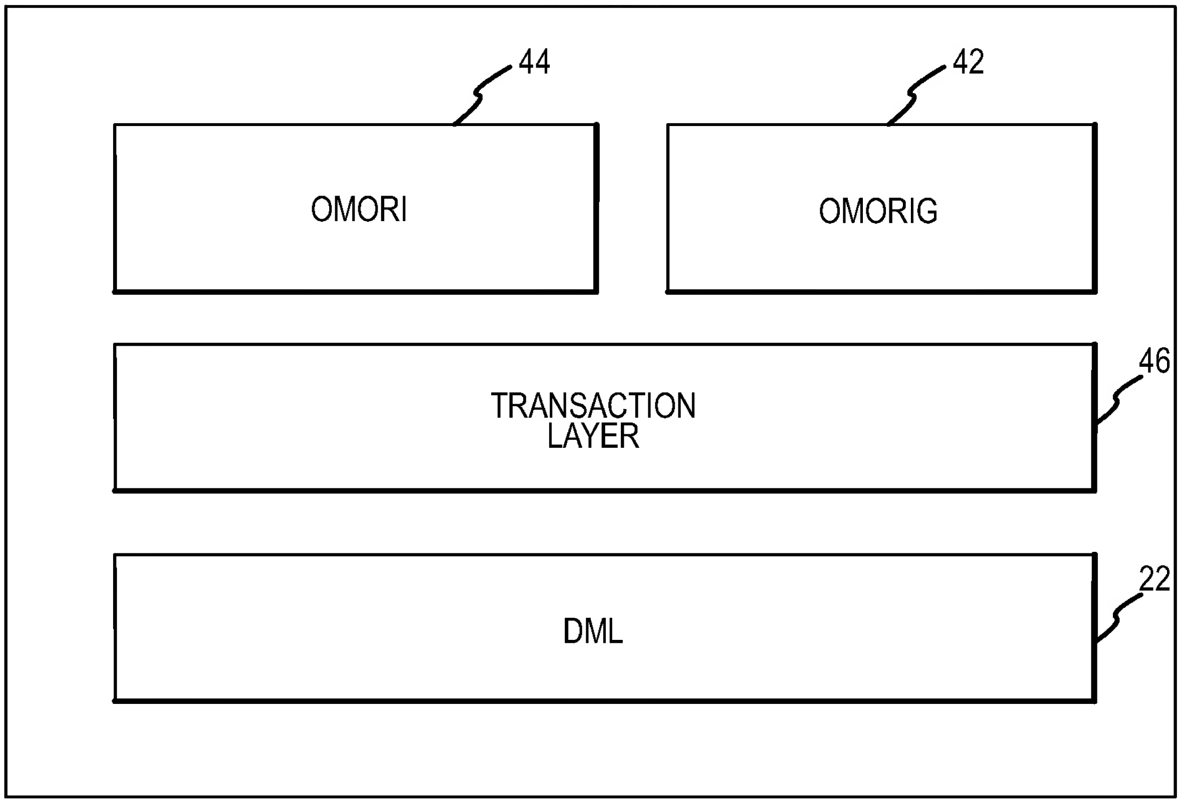
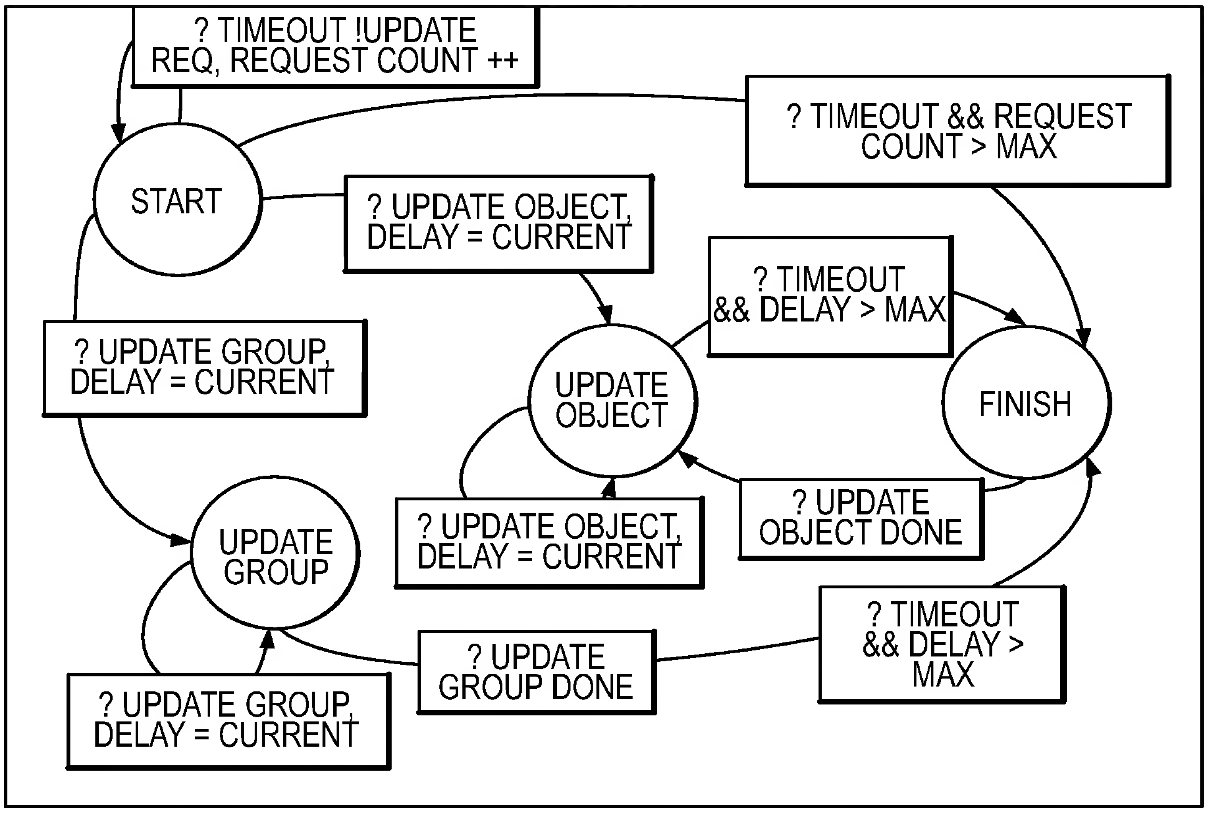


FIG.10


? TIMEOUT !UPDATE REQ, REQUEST COUNT ++
? TIMEOUT && REQUEST COUNT > MAX
START
? UPDATE OBJECT, DELAY = CURRENT
? TIMEOUT && DELAY > MAX
UPDATE OBJECT
FINISH
? UPDATE GROUP, DELAY = CURRENT
? UPDATE OBJECT, DELAY = CURRENT
? UPDATE OBJECT DONE
UPDATE GROUP
? TIMEOUT && DELAY > MAX
? UPDATE GROUP DONE
? UPDATE GROUP, DELAY = CURRENT

STATE	EVENT	ACTION
START	TIMEOUT && (REQUEST COUNT < MAX)	SEND UPDATE REQUEST
START	TIMEOUT && (REQUEST COUNT > MAX)	PEER DID NOT REPLY. UPDATE FAILED TRANSIT TO FINISH STATE.
START	RECV UPDATE GROUP MESSAGE	TRANSIT TO UPDATE GROUP STATE. SET LAST UPDATE EQUAL TO THE CURRENT TIME.
START	RECV UPDATE OBJECT MESSAGE	TRANSIT TO UPDATE OBJECT STATE. SET LAST UPDATE EQUAL TO THE CURRENT TIME.

FIG.11

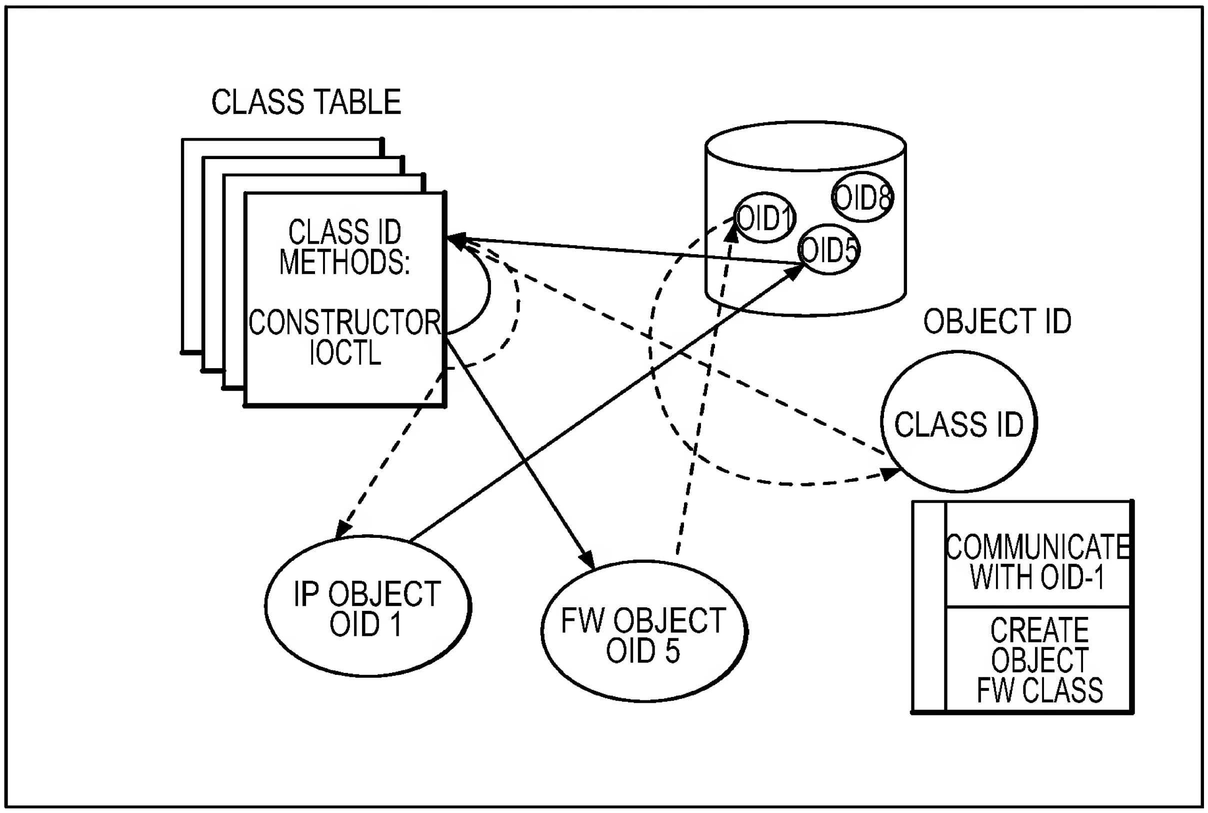

CLASS TABLE
CLASS ID
METHODS:
CONSTRUCTOR
IOCTL
OID1
OID8
OID5
OBJECT ID
CLASS ID
COMMUNICATE WITH OID-1
CREATE OBJECT FW CLASS
IP OBJECT OID 1
FW OBJECT OID 5

FIG.12

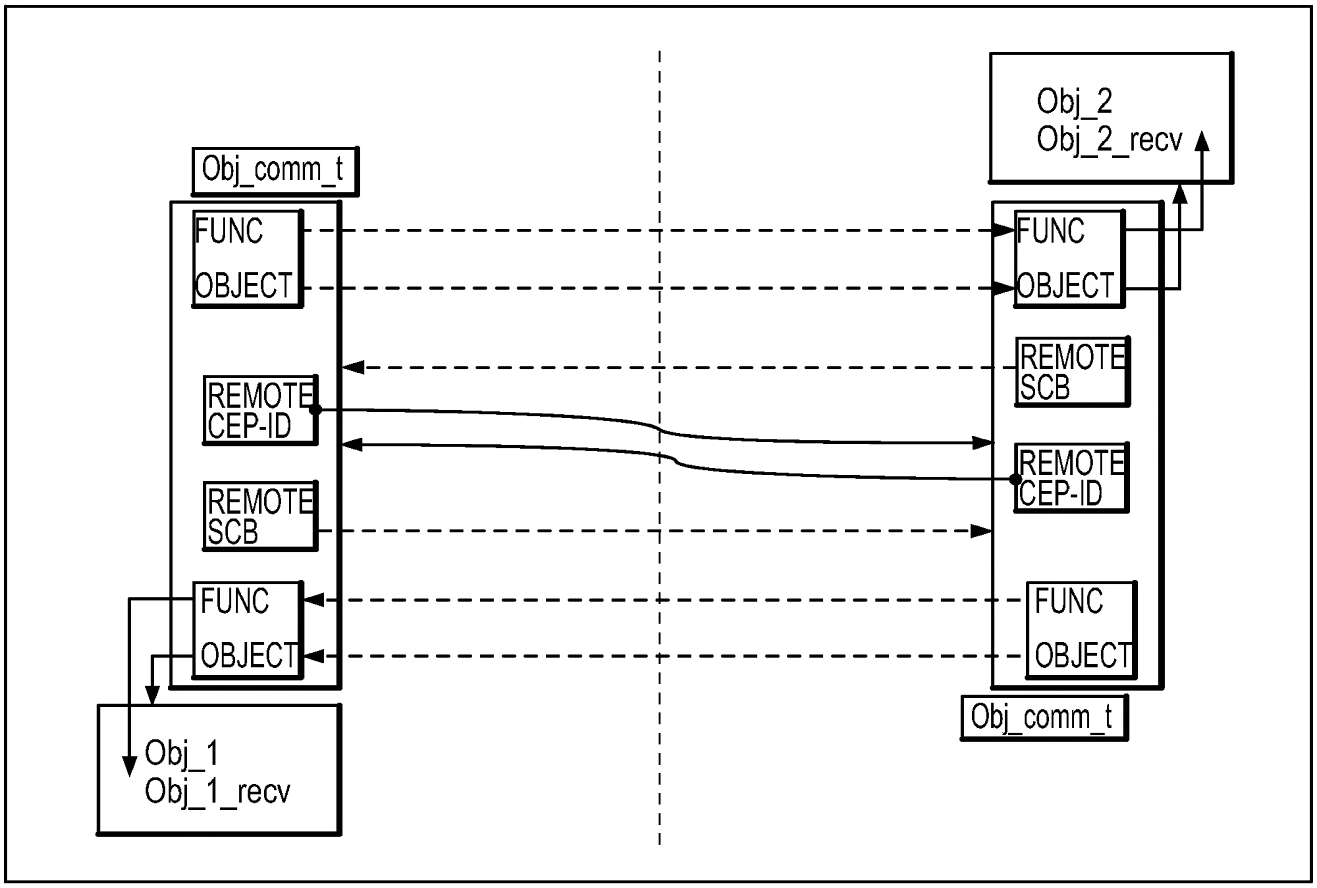

Obj_2
Obj_2_recv
Obj_comm_t
FUNC
OBJECT
REMOTE
CEP-ID
REMOTE
SCB
FUNC
OBJECT
REMOTE
SCB
REMOTE
CEP-ID
FUNC
OBJECT
Obj_comm_t
Obj_1
Obj_1_recv

FIG.13

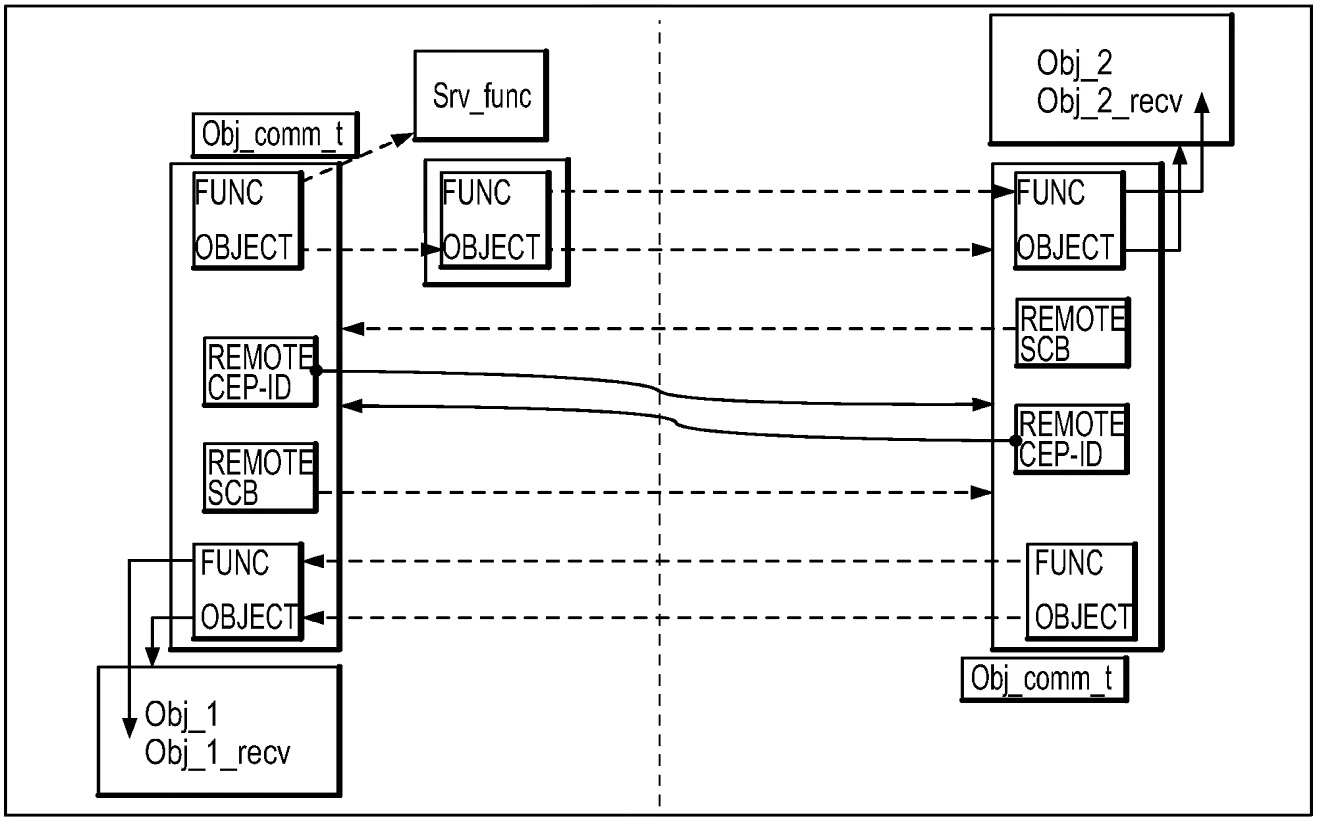

Srv_func
Obj_2
Obj_2_recv
Obj_comm_t
FUNC
OBJECT
FUNC
OBJECT
FUNC
OBJECT
REMOTE
SCB
REMOTE
CEP-ID
REMOTE
CEP-ID
REMOTE
SCB
FUNC
OBJECT
FUNC
OBJECT
Obj_comm_t
Obj_1
Obj_1_recv

FIG.14

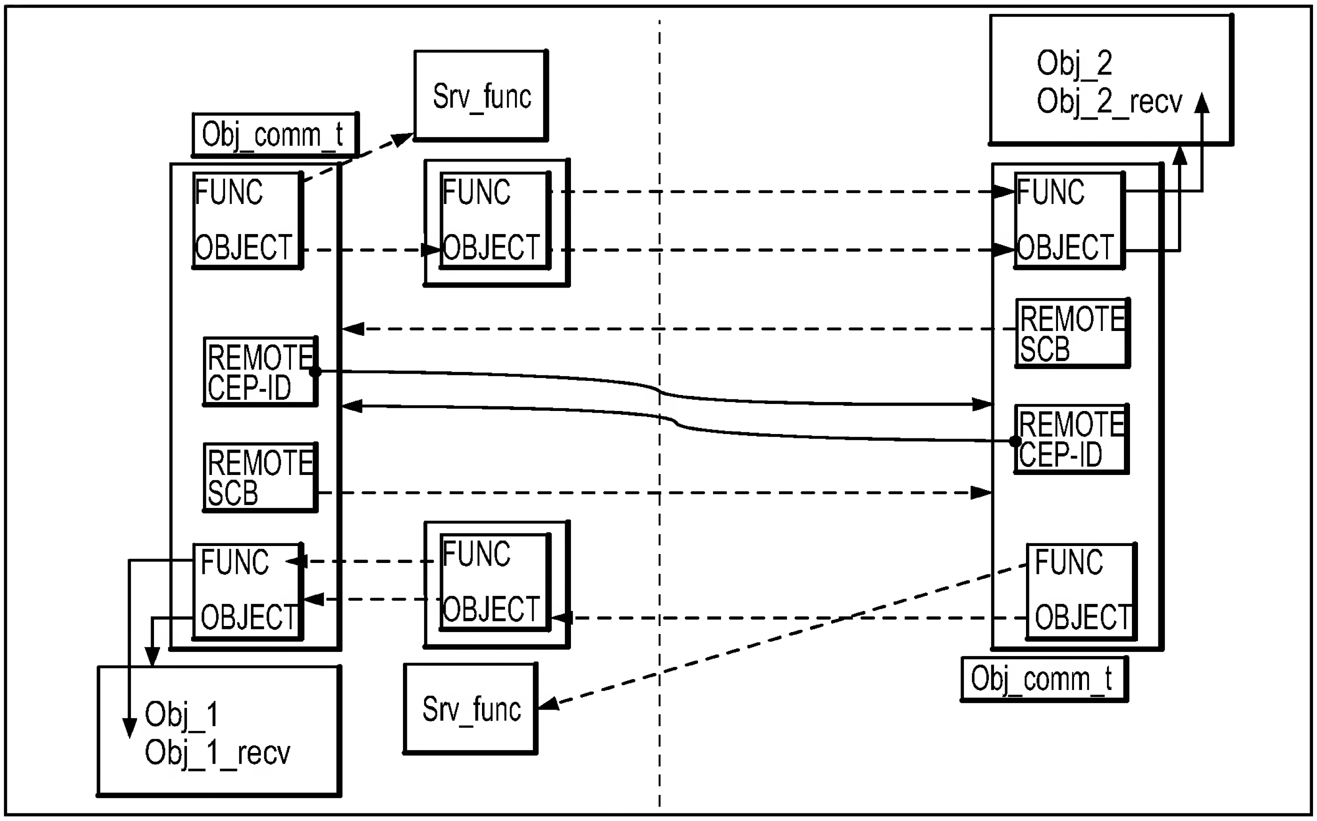

Srv_func
Obj_2
Obj_2_recv
Obj_comm_t
FUNC
OBJECT
FUNC
OBJECT
FUNC
OBJECT
REMOTE
SCB
REMOTE
CEP-ID
REMOTE
CEP-ID
REMOTE
SCB
FUNC
OBJECT
FUNC
OBJECT
FUNC
OBJECT
Obj_comm_t
Obj_1
Obj_1_recv
Srv_func

FIG.15

STEP	LOCAL CEP OBJECT	LOCAL IPNOS	LOCAL RM/LQ	REMOTE RM/LQ	REMOTE IPNOS	REMOTE CEP OBJECT
1	obj_associate_channel(local_chan, local_cep_id, remote_cep_id)					
2		*/* Allocate remote LQ */* resmng_alloc_resource (RESOURCE_DATA_CONNECTION, 0, remote_cep_id -> object.address_space_id, &remote_lq)				
3				Lookup resource tag and allocate from *remote* LQ		
4		*/* Ask remote LQ to allocate local LQ*/* status = omori_obj_ioctl_by_id (&remote_lq, remote_lq.group, OBJ_CTL_CODE_ANY (LQUSER_BIND), &lq_bind, sizeof (lq_bind)); memcpy (&local_lq,				

FIG.16a

		&lq_bind.lq_object.local, sizeof (object_id_t));				
5				Use resmng_alloc_resource0 to allocate *local* LQ		
6			Lookup resource tag and allocate from *local* LQ			
7				Return allocated *local* LQ		
8		*/* Bind Local and Remote LQs*/* status = omori_obj_ioctl_by_id (&local_lq, local_lq.group, OBJ_CTL_CODE_ANY (LQUSER_BIND), &lq_bind, sizeof (lq_bind));				
9			Setup LQ-API parameters to point to *remote* LQ	Setup LQ-API parameters to point to *local* LQ		
10		*/* Push local LQ as a service onto local channel*/* status = omori_obj_ioctl_by_id				

FIG.16b

		(&local_lq, local_lq.group, OBJ_CTL_CODE_ANY (LQUSER_BIND), &lq_bind, sizeof (lq_bind));				
11	Lookup CEP address					
12					*/* Push remote LQ as a service onto remote channel*/* status = omori_obj_ioctl_by_id (&remote_lq, local_lq.group, OBJ_CTL_CODE_ANY (LQUSER_BIND), &lq_bind, sizeof (lq_bind));	
13						Lookup CEP address

FIG.16c

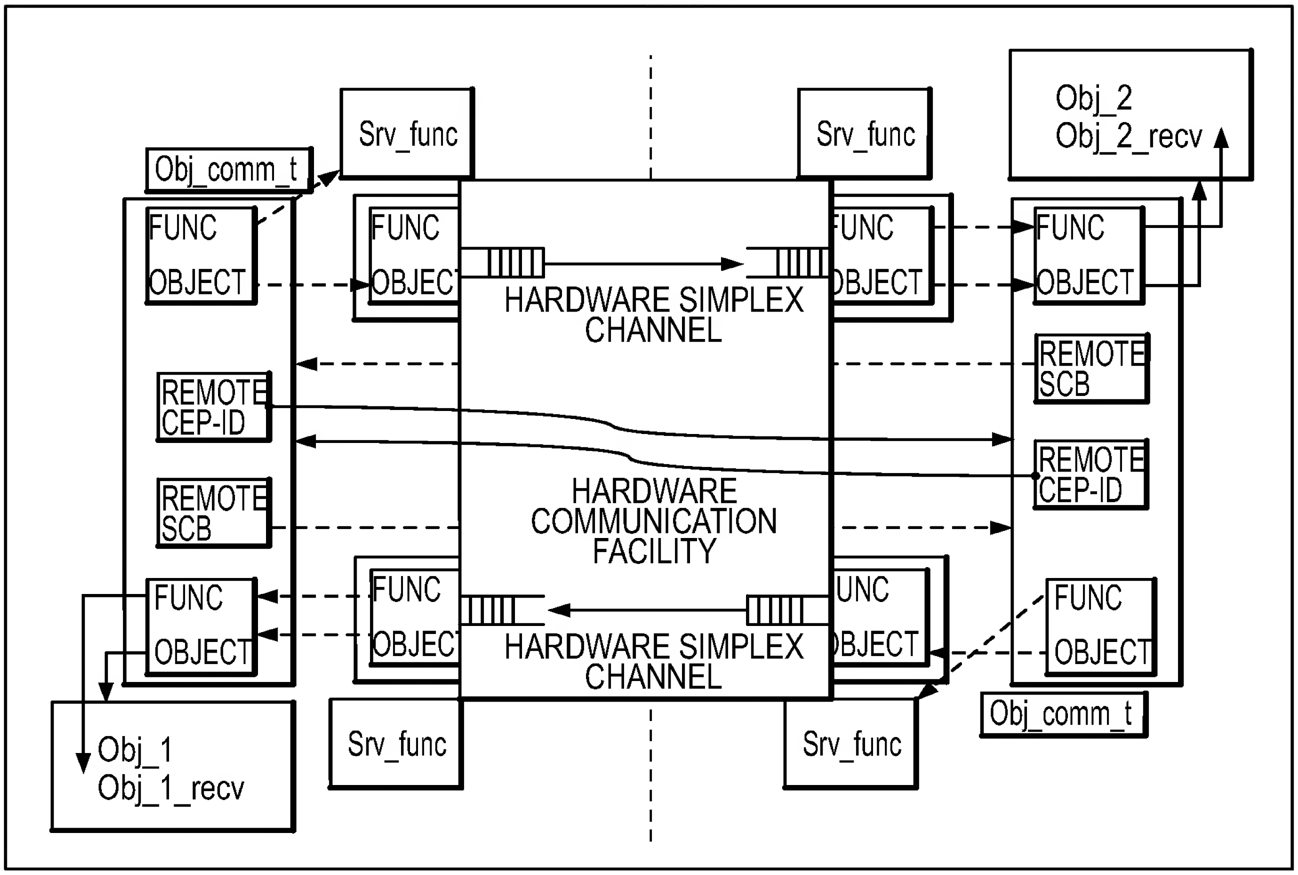

Srv_func
Srv_func
Obj_2
Obj_2_recv
Obj_comm_t
FUNC
OBJECT
FUNC
OBJECT
FUNC
OBJECT
FUNC
OBJECT
HARDWARE SIMPLEX CHANNEL
REMOTE SCB
REMOTE CEP-ID
REMOTE CEP-ID
REMOTE SCB
HARDWARE COMMUNICATION FACILITY
FUNC
OBJECT
FUNC
OBJECT
FUNC
OBJECT
FUNC
OBJECT
HARDWARE SIMPLEX CHANNEL
Obj_1
Obj_1_recv
Srv_func
Srv_func
Obj_comm_t

FIG.17

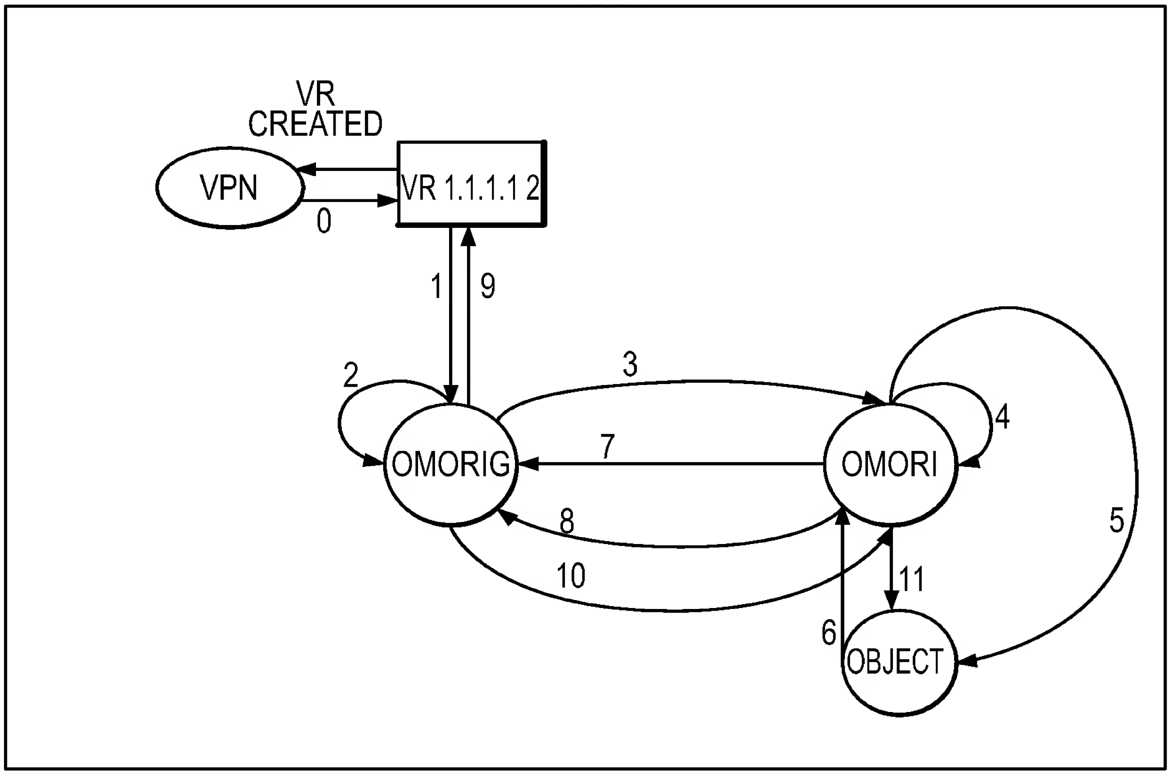

VR CREATED
VPN
VR 1.1.1.1 2
0
1
9
2
3
4
OMORIG
7
OMORI
8
5
10
11
6
OBJECT

FIG.18

STEP	OMCD	OMORIG	OMORI	OBJECT
0	Create unique vr_descriptor_t for specified VPN, fills with default value and adds VR to the list of VR per VPN.			
1	Requests group creation for specified VR with class_group_selector on specified address space.			

FIG.19a

2		Create group 1; create object id link of selected class. Validate address space id on capability to service specified object class. Send request CREATE_OBJECT to capable OMORI (2). Wait for OMORI reply.		
3			Receive CREATE_OBJECT request for specified group. Lookup a group; not found. Create group 1; Create object descriptor of selected class. Call the class constructor.	
4		Receive MV_OBJ_TO_GROUP request; add object id to OMORIG Database	add object to the group, send MV_OBJ_TO_GROU P request to OMORIG	
5				Create and initialize an object; return SUCCESS or FAILURE.
6			If FAILURE remove object from the group, send MV_OBJ_TO_GROU P and RM_OBJ_FROM_GR OUP to OMORIG; ================= Else send reply for CREATE_OBJECT request to OMORIG	
7		Receive MV_OBJ_TO_GROUP request, move object to the group 0(OM_BASE_GROUP); Receive RM_OBJ_FROM_GROU P request; remove object id from OMORIG		

FIG.19b

		Database =================		
8		Receive Object CREATE reply. Signal to OMCD		
9	VR created, return status to user			
10		Send ACTIVATE object message to OMORI (2)		
11			Receive ACTIVATE object message. For all the objects of this group send generic IOCTL ACTIVATE_OBJECT	Activate object, Do object-specific action to make it active, operational

FIG.19c

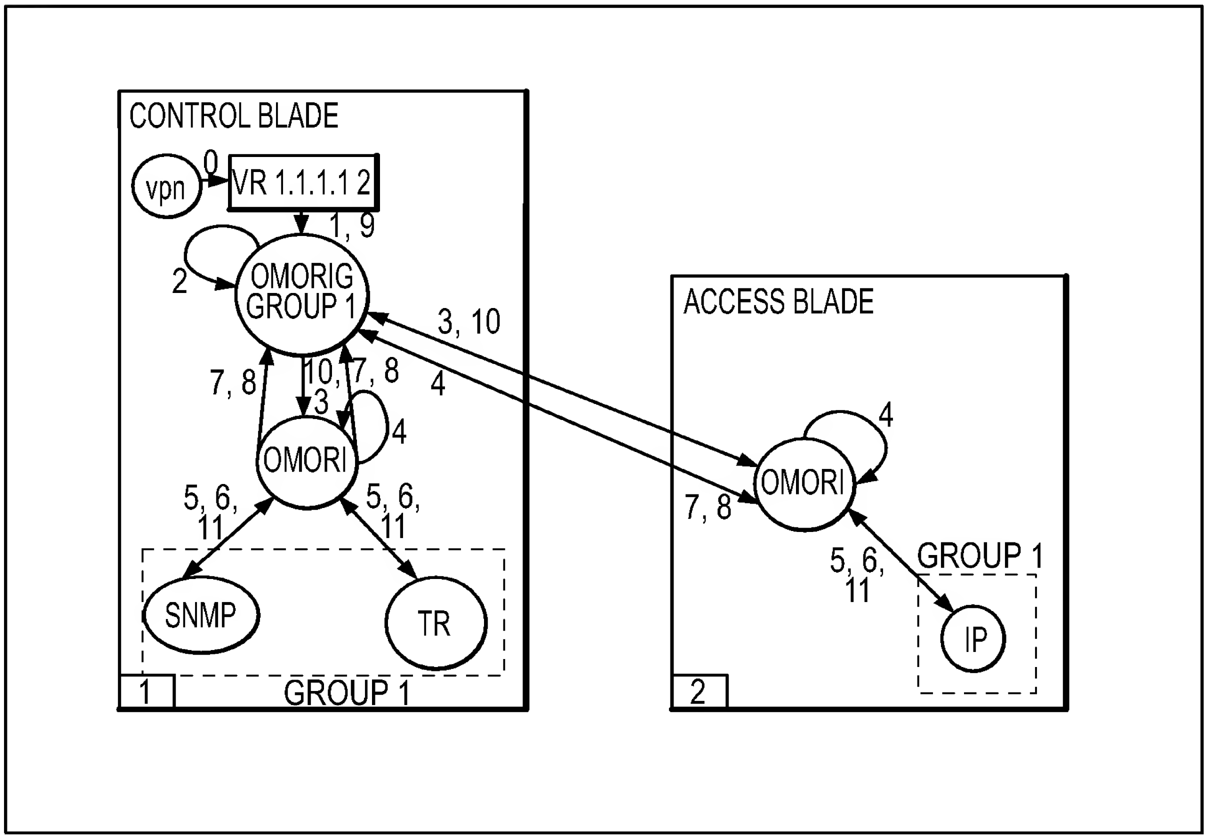


FIG.20

STEP	OMCD	OMORIG	OMORI	OBJECT
0	Create unique vr_descriptor_t for specified VPN, fills with default value and adds VR to the list of VR per VPN.			
1	Requests group creation for specified VR with class_group_selector on specified address space.			
2		Create group 1; create object id link of selected class. Validate address space id on capability to service specified object class. Send request CREATE_OBJECT to capable OMORIs (1 and 2). Wait for reply from both OMORIs.		
3			Receive CREATE_OBJECT request for specified group. Lookup a group; not found. Create group 1; Create object descriptor of selected class. Call the class constructor.	
4		Receive MV_OBJ_TO_GROUP request; add object id to OMORIG Database	add object to the group, send MV_OBJ_TO_GROUP request to OMORIG	
5				Create and initialize an object; return SUCCESS or FAILURE
6			If FAILURE remove object from the group, send MV_OBJ_TO_GROUP and RM_OBJ_FROM_GROUP to OMORIG;	

FIG.21a

			Else send reply for CREATE_OBJECT request to OMORIG	
7		Receive MV_OBJ_TO_GROUP request, move object to the group 0(OM_BASE_GROUP); Receive RM_OBJ_FROM_GROUP request; remove object id from OMORIG Database		
8		Receive Object CREATE reply. If all the object replied then signal to OMCD, otherwise do nothing		
8	VR created, return status to user			
10		Send ACTIVATE object message to every OMORI (1,2) where objects were created		
11			Receive ACTIVATE object message. For all the objects of this group send generic IOCTL ACTIVATE_OBJECT	
12				Activate object, Do object-specific action to make it active, operational

FIG.21b

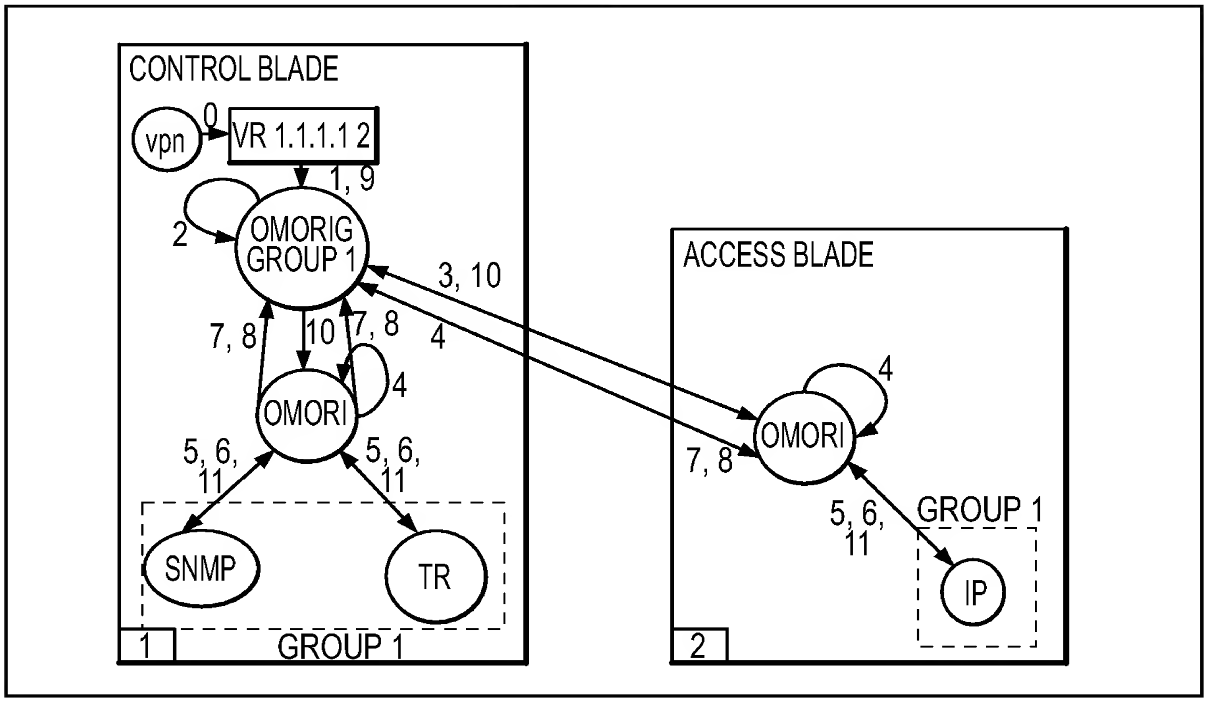


FIG.22

STEP	OMCD	OMORIG	OMORI	OBJECT
0	For specified VPN and VR lookup vr_descriptor. Call OMORIG to delete corresponding group.			
1		Lookup group descriptor by specified id. Filter OMORIs which have objects to be destroyed(which belong to the specified group)		
2			Receive DESTROY_GROUP_OBJECTS request for specified group. Lookup a group; Send generic IOCTL STOP_OBJECT to every local object, which belongs to the group	
3				Stop operating, destroy all connections with other objects

FIG.23a

4			Send generic IOCTL DESTROY_OBJECT to every local object, which belongs to the group	
5				Free itself
6			If FAILURE remove object from the group, send MV_OBJ_TO_GROUP and RM_OBJ_FROM_GROUP to OMORIG;	
7		Receive MV_OBJ_TO_GROUP request, move object to the group 0(OM_BASE_GROUP); Receive RM_OBJ_FROM_GROUP request; remove object id from OMORIG Database		
8		Receive Object DESTROY_GROUP_OBJECTS. Subtract number of destroyed objects from the total number of objects in the group (VR). If all objects destroyed then signal to OMCD, otherwise do nothing.		
8	VR destroyed, return status to user			

FIG.23b

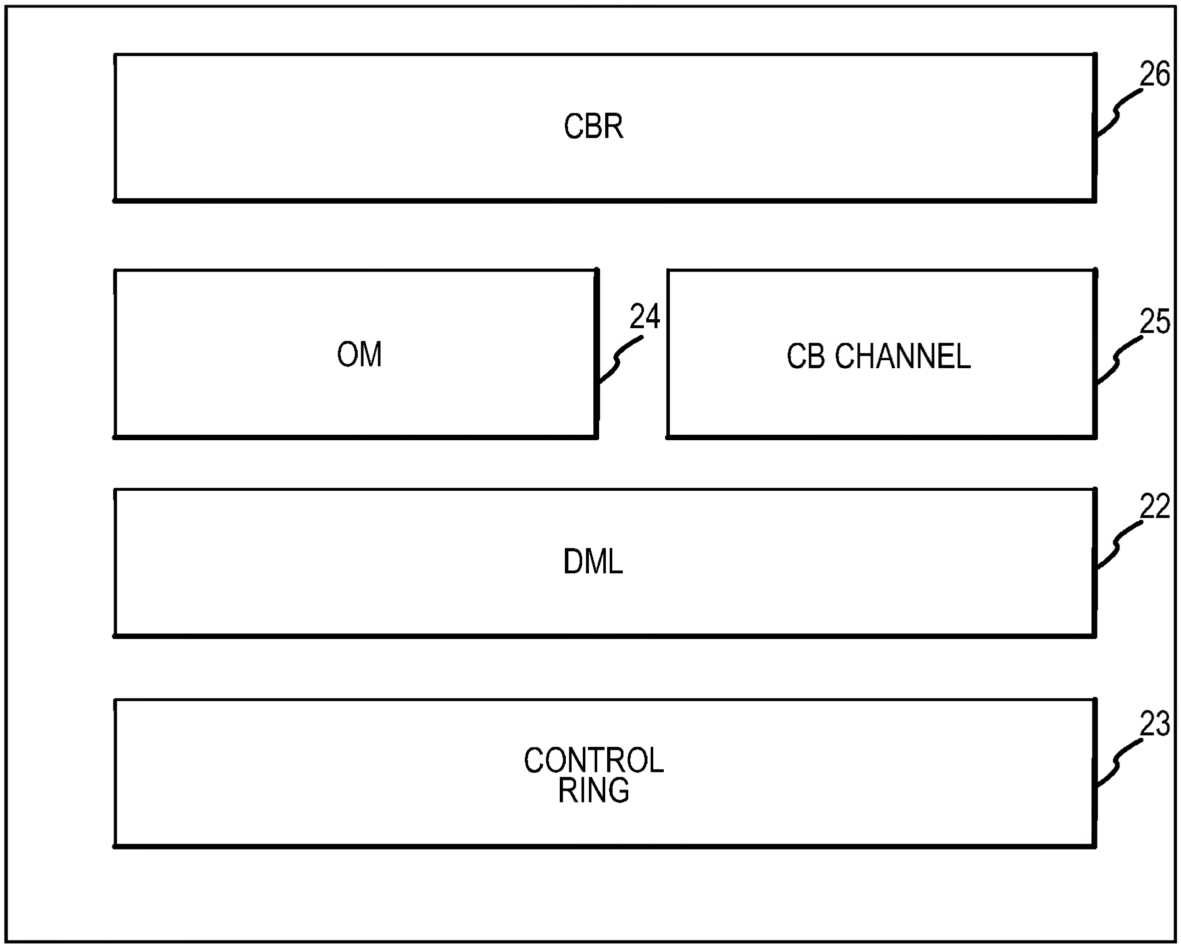


FIG.24

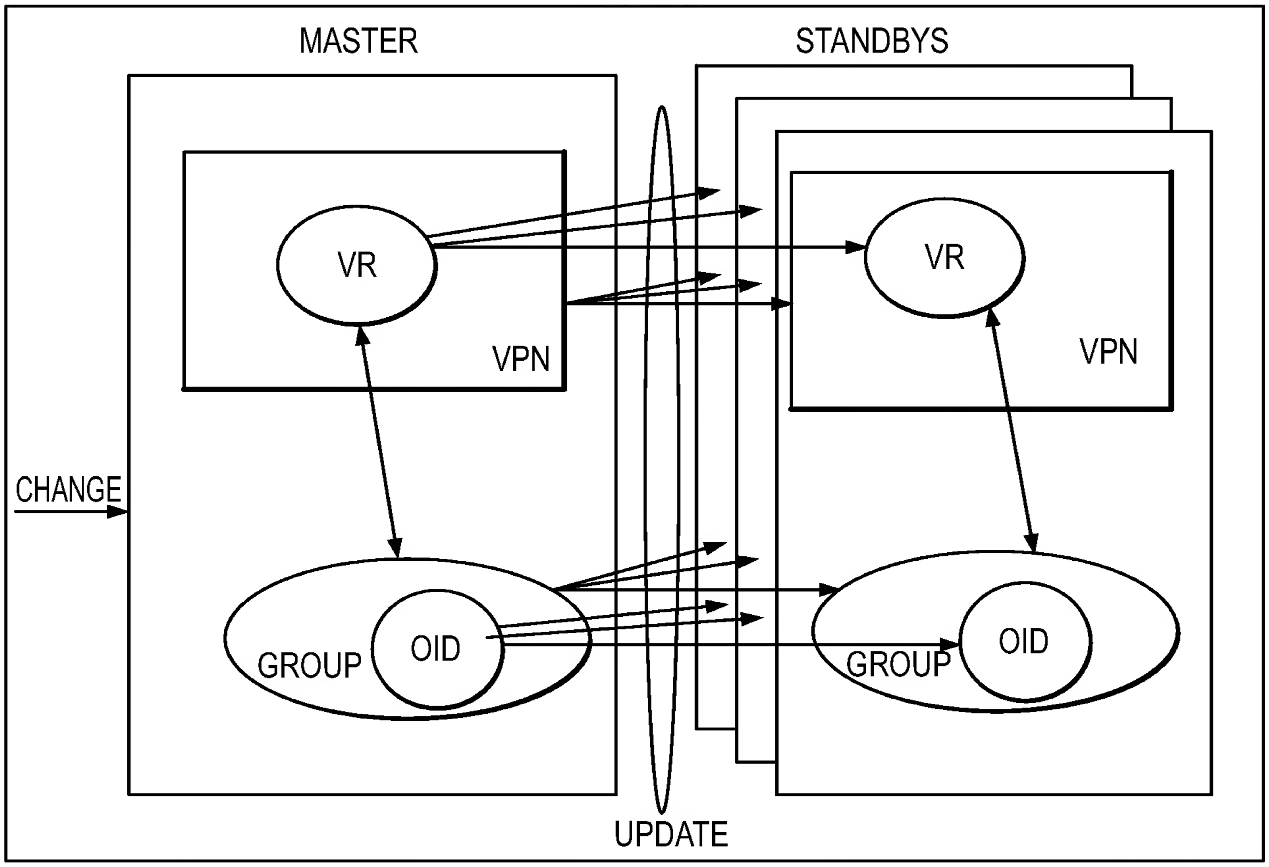

MASTER
STANDBYS
VR
VPN
VR
VPN
CHANGE
GROUP
OID
GROUP
OID
UPDATE

FIG.25

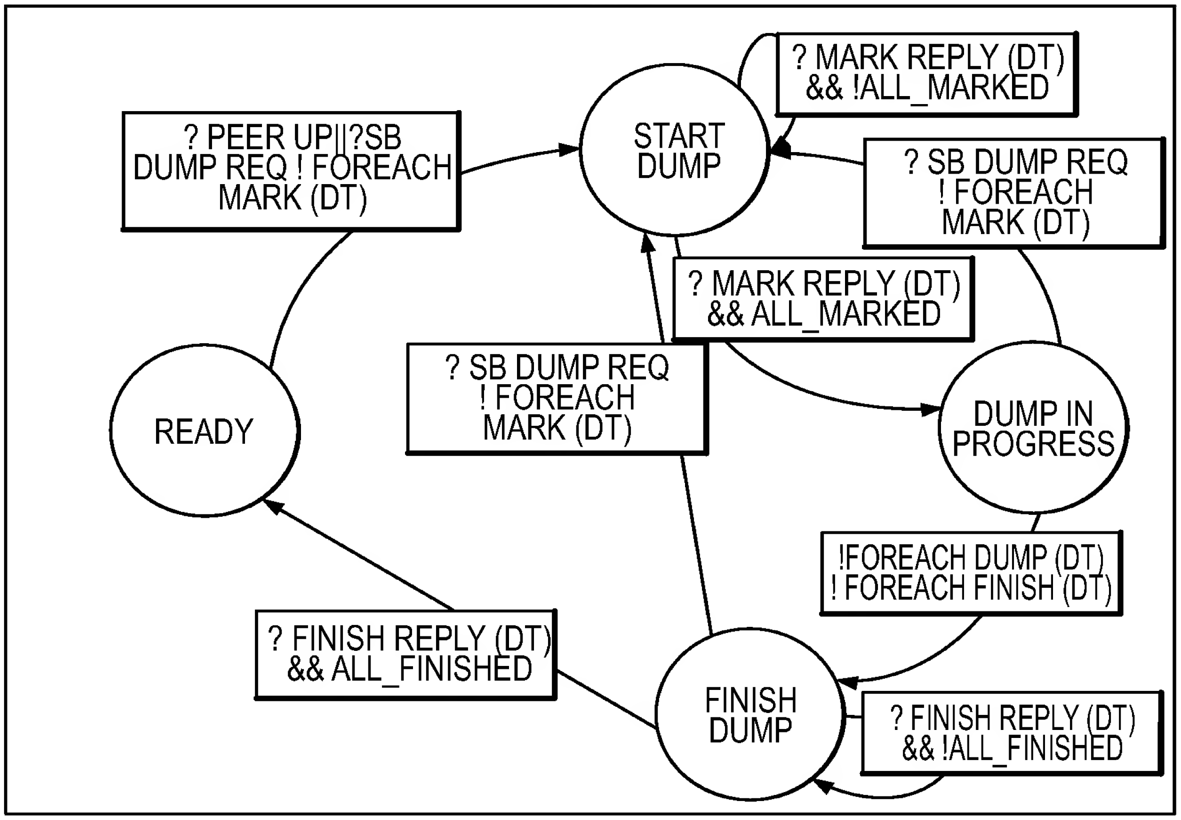

? PEER UP||?SB DUMP REQ ! FOREACH MARK (DT)
START DUMP
? MARK REPLY (DT) && !ALL_MARKED
? SB DUMP REQ ! FOREACH MARK (DT)
? MARK REPLY (DT) && ALL_MARKED
READY
? SB DUMP REQ ! FOREACH MARK (DT)
DUMP IN PROGRESS
!FOREACH DUMP (DT) ! FOREACH FINISH (DT)
? FINISH REPLY (DT) && ALL_FINISHED
FINISH DUMP
? FINISH REPLY (DT) && !ALL_FINISHED

FIG.26

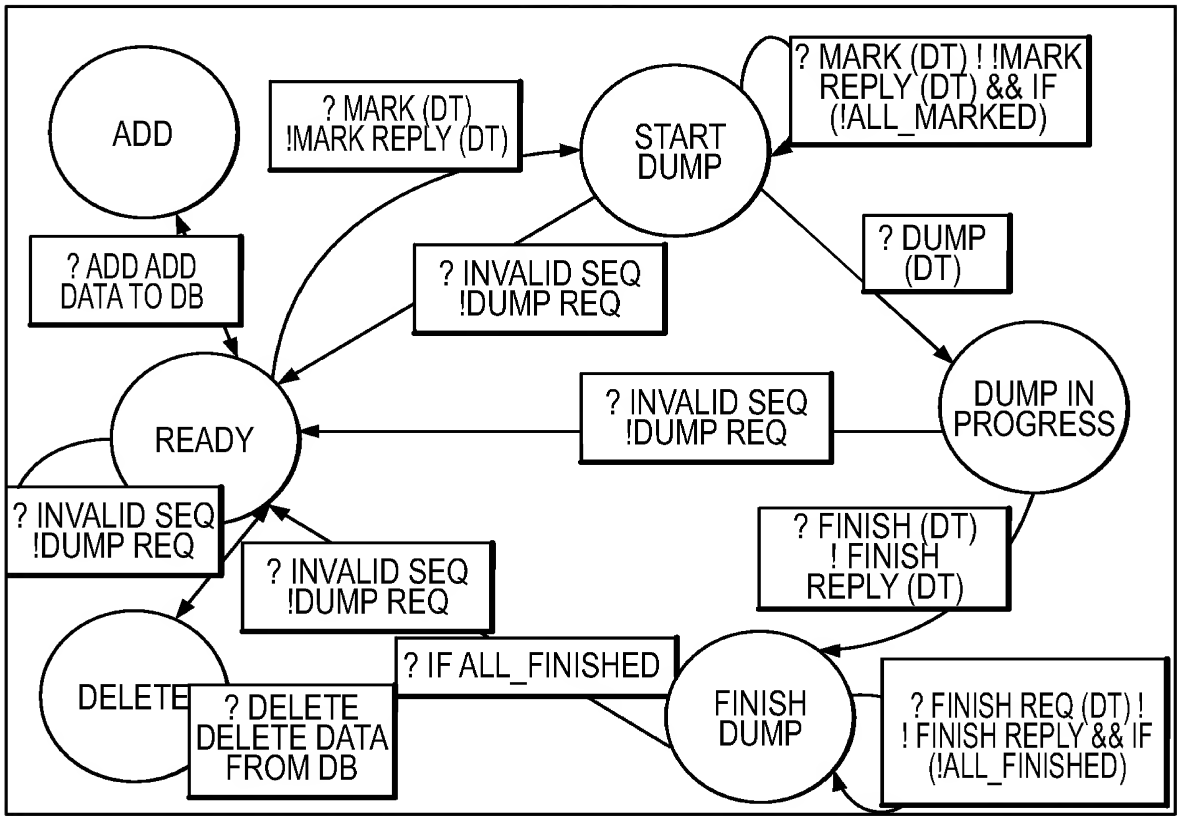

ADD
? MARK (DT) !MARK REPLY (DT)
START DUMP
? MARK (DT) ! !MARK REPLY (DT) && IF (!ALL_MARKED)
? ADD ADD DATA TO DB
? INVALID SEQ !DUMP REQ
? DUMP (DT)
READY
? INVALID SEQ !DUMP REQ
DUMP IN PROGRESS
? INVALID SEQ !DUMP REQ
? FINISH (DT) ! FINISH REPLY (DT)
? INVALID SEQ !DUMP REQ
? IF ALL_FINISHED
DELETE
? DELETE DELETE DATA FROM DB
FINISH DUMP
? FINISH REQ (DT) ! ! FINISH REPLY && IF (!ALL_FINISHED)

FIG.27